

最佳TypeScript教程

黑马程序员深圳校区 前端教研部

本周四、周五：18:30 - 20:30



自我介绍



编程车上不吃亏，黑马老邹带你飞

黑马老邹

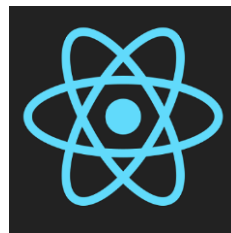
有一点点坚定的理想主义者

TypeScript到底要不要学？

大局观

越早学习，你在未来将越加有**实力**立于不败之地！

TypeScript是前端的**未来**！



TypeScript

Weekly Downloads

8,482,936





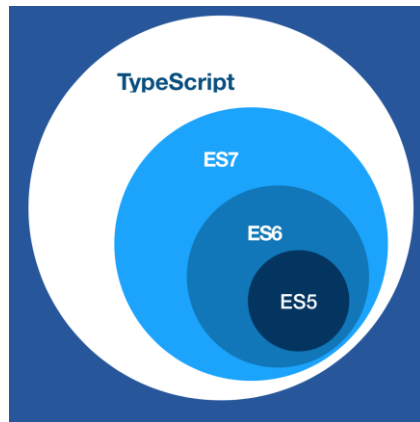


TypeScript 是微软推出的一门语言

TypeScript 是 JavaScript 的超集，包含 ES

新增了类型系统和完整的面向对象语法

使用 TS 编写的项目更健硕，且更容易扩展和维护



我们的理想：为社会做更多贡献~~~

1

TypeScript基础

2

'真'面向对象

3

TypeScript 与 Vue 的结合

vue/typescript 脚手架
vuex
vue-cli

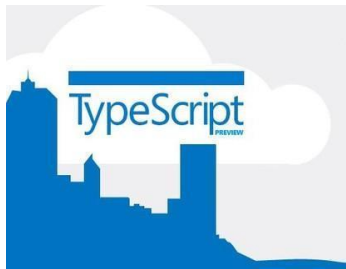
4

综合项目 - 黑马Memo便签





课程安排



Day01.TypeScript基础 和 面向对象

Day02.TypeScript+Vue 与 黑马便签

第一章 TypeScript 基础

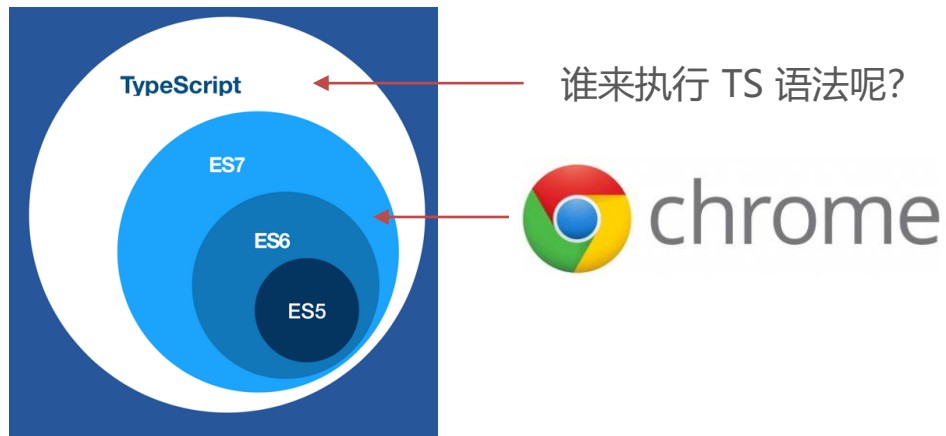
第一章 TypeScript 基础

01. TypeScript 环境安装与运行

02. TypeScript 变量与数据类型

03. TypeScript 函数

01. TypeScript 环境安装与运行



TS -> JS , 再交给浏览器运行. 类似 Less -> CSS.

01. TypeScript 环境安装与运行

第1步：安装 TypeScript 之前 先要 安装 NodeJS 和 NPM

第2步：通过 npm 安装 TypeScript



<https://nodejs.org>

node 是独立于浏览器运行的 js 环境

npm 会随着 node 一起被安装



<https://npmjs.com/>

nodejs 包管理器

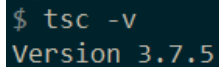
通过npm可以安装各类工具，尤其是前端开发工具

node -v 和 npm -v 分别 校验是否安装成功

01. TypeScript 环境安装与运行

全局安装 typescript: `npm install -g typescript`

校验 typescript: `tsc -v`



```
$ tsc -v
Version 3.7.5
```

tsc 作用: 负责将 ts 代码 转为 浏览器 和 nodejs 识别的 js 代码

01. TypeScript 环境安装与运行 | 运行

01. 在后缀名为.ts的文件中书写 typescript 代码
02. 使用 tsc 将 typescript 代码编译 js 代码
03. 在浏览器或者 nodejs 中执行 js 代码



01. TypeScript 环境安装与运行 | 运行

- 01hello.ts

```
let str: string = 'Ruiky';  
console.log('hello ' + str);
```

编译



- 01hello.js

```
var str = 'Ruiky';  
console.log('hello' + str);
```

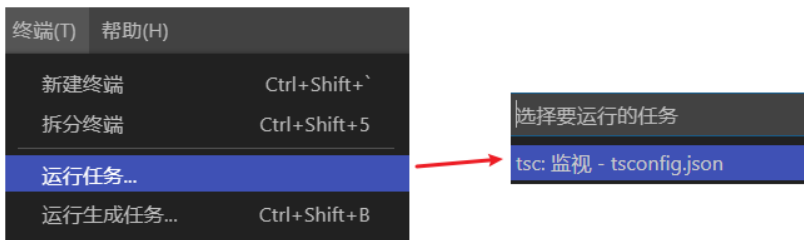
- 将 js 文件 引入 html 文件使用

01. TypeScript 环境安装与运行 | 自动编译

自动编译：就是省去程序员敲击命令编译文件，由工具来自动完成

设置 VSCode 自动编译

01. 运行 `tsc --init`，创建 `tsconfig.json` 文件
02. 修改 `tsconfig.json` 文件，设置 `js` 文件夹：`"outDir": "./js/"`
03. 设置 `vscode` 监视任务：



小结

01. TypeScript 环境安装与运行

- ◆ 下载安装：使用 npm 下载安装 TS 环境
- ◆ 编译 TS 文件 -> JS 文件 -> 在 html 文件中使用
- ◆ 设置 vscode 自动 编译 TS 文件

第一章 TypeScript 基础

01. TypeScript 环境安装与运行

02. TypeScript 变量与数据类型

03. TypeScript 函数

02. TypeScript 变量与数据类型 | 变量

- 变量 语法

JavaScript

```
let 变量名 = 值;
```

```
let zouAge = 18;
```



TypeScript

```
let 变量名: 变量类型 = 值;
```

```
let zouAge:number = 18;
```

- 在 TS 中，为变量指定了类型，就只能给这个变量设置相同类型的值

```
let dName: string = 'Ruikey' ;
```

```
dName = 520;
```

报错: Type '520' is not assignable to type 'string'

02. TypeScript 变量与数据类型 | 数据类型

- 原有类型

string	number	boolean	Array
null	undefined	Symbol	Object

- 新增类型

tuple 元组	enum 枚举	
any 任意类型	never	void

02. TypeScript 数据类型 | 数组

需要声明时指定 数组中元素的类型

- 语法:

方式一: `let` 数组名: `类型[]` = [`值1`, `值2`];

```
let arrHeros: string[] = [ '安琪拉' , '亚索' , '大乔' ];
```

方式二: `let` 数组名: `Array<类型>` = [`值1`, `值2`];

```
let arrHeros: Array<string> = [ '安琪拉' , '亚索' , '大乔' ];
```

- 特点:

元素类型 固定

长度不限制

02. TypeScript 数据类型 | 元组

- **概念：** 就是一个规定了元素数量 和 每个元素类型的"数组"
而每个元素的类型，可以不相同

- **语法：**

let 元组名: [类型1, 类型2, 类型3] = [值1,值2 ,值3];

```
let tup1: [string, number, boolean] = ['讨厌~~', 18, true];
```

- **为什么要有元组？** TS中数组元素类型必须一致，如需要不同元素，可以用 **元组** 了！

- **特点：**

声明时，要指定元素个数

声明时，要为每个元素规定 类型

02. TypeScript 数据类型 | 枚举

- 问题：性别标识



- 声明语法：

```
enum 枚举名 {  
    枚举项1 = 枚举值1,  
    枚举项2 = 枚举值2,  
    ...  
}
```

eg. →

```
enum GunType {  
    M416 = 1,  
    AK47 = 2,  
    Goza = 3  
}
```

枚举项 一般用英文和数字，而 枚举值 用整型数字

- 使用默认枚举值：

```
enum 枚举名 {  
    枚举项1,  
    枚举项2,  
    .....  
}
```

eg. →

```
enum GunType {  
    M416, // -> 0  
    AK47, // -> 1  
    Goza  // -> 2  
}
```

枚举值 将自动生成从 0 开始的数值

02. TypeScript 数据类型 | 枚举

- 问题：性别标识



- 解决问题：

```
// 声明性别枚举
enum Gender{
    Boy = 1,
    Girl = 2,
    Unknown = 3
}
```

```
// 创建 用户性别变量
let usrSex: Gender = Gender.Boy;

// 判断 变量中的性别是否为 Boy
if(usrSex == Gender.Boy){
    console.log(usrSex); // 1
}else{
    console.log(usrSex); // 2 or 3
}
```


02. TypeScript 变量与数据类型 | 数据类型

- 原有类型

string	number	boolean	Array
null	undefined	Symbol	Object

- 新增类型

tuple 元组	enum 枚举	
any 任意类型	never	void

02. TypeScript 数据类型 | any

- **概念：** any 代表任意类型，一般在获取 dom 时使用

我们在接收用户输入 或 第三方代码库时，还不能确定会返回什么类型的值，此时也可以使用 any类型

- **示例：**

```
let txtName: any = document.getElementById('txtN');
```

02. TypeScript 数据类型 | void

- 概念：void 代表没有类型，一般用在无返回值的函数
- 语法：

```
function sayHi1(): string {  
    return 'hi,你好呀~~';  
}
```



```
let re1 = sayHi1();
```

```
function sayHi2(): void {  
    console.log('hi啥，讨厌，死鬼~~~');  
}
```



```
sayHi2();
```

02. TypeScript 数据类型 | never

- 概念：never 代表不存在的值的类型，常用作抛出异常或无限循环的 函数返回类型
- 语法：

```
function test():never{  
    while(true){  
  
    }  
}
```

```
function test2():never{  
    throw new Error('讨厌，死鬼~');  
}
```

- 补充：never类型是ts中的底部类型，所有类型都是never类型的父类
所以 never类型值可以赋给任意类型的变量

```
let x:never = test();  
let y:string = test();
```

02. TypeScript 数据类型 | 类型推断

- 概念：如果变量的声明和初始化是在同一行，可以省略掉变量类型的声明。

TypeScript

```
let 变量名 = 值;
```

相当于

```
let 变量名: 变量类型 = 值;
```

- 验证：

```
let age = 18;
```

 此时变量 age 的类型被推断为 number

```
age = 'jack';
```

 报错，因为变量 age 的类型是 number

02. TypeScript 数据类型 | 联合类型

- **概念**: 表示取值可以为多种类型中的一种

```
let 变量名: 变量类型1 | 变量类型2 = 值 ;
```

- eg. 接收 prompt 函数的返回值

```
let dName: string | null = prompt('请输入小狗狗名字:');  
console.log('hello' + dName);
```

小结

02. TypeScript 变量与数据类型

- ◆ 掌握了几个原有数据类型
- ◆ 掌握了几个新增数据类型
- ◆ 掌握了 类型推断 语法
- ◆ 掌握了 联合类型 语法

Array

```
// JS语法 创建数组
let arrJS = [1, 'a', true, [], {}];
// TS语法 创建数组
let arrHero: string[] = ['亚索', '安琪拉', '大乔'];
let arrHeroAge: number[] = [18, 21, 27];
// 泛型语法 创建数组
let arrHeroAge2: Array<number> = [18, 21, 27];
```

```
let dName: string | null = prompt('请输入小狗狗名字:');
```

第一章 TypeScript 基础

01. TypeScript 环境安装与运行

02. TypeScript 变量与数据类型

03. TypeScript 函数

03. TypeScript 函数 | 返回值和参数

- 函数 返回值类型

如果函数没有返回值，则定义为 void

```
function 函数名():返回值类型 {  
  
}  
let 变量名:变量类型 = 函数名();
```

- 函数 形参类型

```
function 函数名(形参1:类型 , 形参2:类型):返回值类型 {  
  
}  
let 变量名:变量类型 = 函数名(实参1, 实参2);
```

- 特点

实参 和 形参 的类型要一致

实参 和 形参 的数量要一致

03. TypeScript 函数 | 返回值和参数

小结

函数必须定义 **返回值类型**，如果没有返回值，则定义返回值类型为 **void**

实参 和 形参 的**类型**要一致

实参 和 形参 的**数量**要一致

03. TypeScript 函数 | 可选参数

- 函数 可选参数

可选参数的实参可传，也可不传

```
function 函数名(形参 ? : 类型):返回值类型 {  
      
}
```

- 调用

可以不传递实参

```
函数名();
```

可以传递实参

```
函数名(实参值);
```

03. TypeScript 函数 | 默认值

- 函数 默认值

```
function 函数名(形参1?:类型,形参2?:类型):返回值类型 {  
  
}
```

03. TypeScript 函数 | 默认值

- 函数 默认值 ~~形参1:类型 = 默认值1~~ 带默认值的参数 本身也是可选参数

```
function 函数名(形参1:类型 = 默认值1, 形参2:类型 = 默认值2):返回值类型 {  
      
}
```

- 调用

不传递实参

函数名();

编译后

函数名(默认值1, 默认值2);

传1个实参

函数名(实参1);

函数名(实参1, 默认值2);

传2个实参

函数名(实参1, 实参2);

函数名(实参1, 实参2);

只传第2个 实参

函数名(undefined , 实参2);

函数名(默认值1, 实参2);

03. TypeScript 函数 | 剩余参数

- 问题
- 数量不确定的参数: c, d, e ...

```
function add(a : number , b : number):void {  
    console.log(a + b);  
}
```

- 函数 剩余参数

```
function add(形参1:类型,形参2:类型, ... 形参3:类型[]):void {  
    console.log(a + b);  
}
```

- 特点

剩余参数 只能 定义有一个

剩余参数 只能 定义为数组

剩余参数 只能 定义在 形参列表最后

- 调用

传递2个实参

```
函数名( 1,2 );
```

传递3个实参

```
函数名( 1, 2, 4 );
```

传递2+个实参

```
函数名( 1, 2, 3, 4, 5, 6, 7 );
```

小结

03. TypeScript 函数

- ◆ 掌握 为函数返回值和参数指定类型
- ◆ 掌握 函数可选参数
- ◆ 掌握 函数默认值
- ◆ 掌握 函数剩余参数

```
function add(形参1: 类型, 形参2: 类型, ... 形参3: 类型[]): void {  
    console.log(a + b);  
}
```

最佳TypeScript教程

黑马程序员深圳校区 前端教研部

第二天：面向对象 和 VueTS案例
18:30 - 20:30



自我介绍



编程车上不吃亏，黑马老邹带你飞

黑马老邹

有一点点坚定的理想主义者

2

核心语法

变量与数据类型

函数

类



2

核心语法

变量与数据类型

函数

类

封装

继承

多态



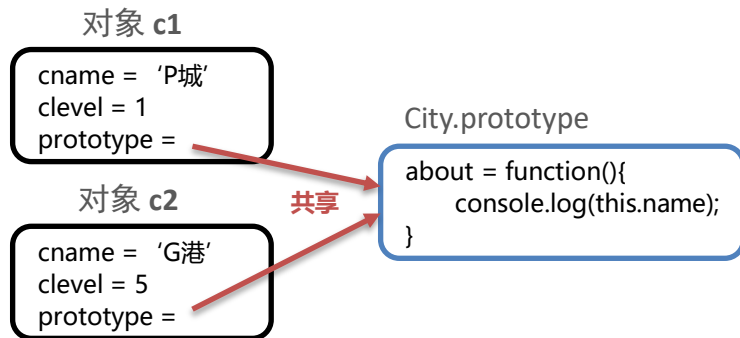
类 – 批量创建对象

- 创建对象 (构造函数+new)

```
function City(cName, cLevel) {  
    this.cname = cName;  
    this.clevel = cLevel;  
}  
this.about = function...  
City.prototype.about = function () {  
    console.log(`兄嘚, 你跳【${this.cname}】~此地危险系数为: 【${this.clevel}】`);  
}
```

- 调用

```
let c1 = new City('P城', 1);  
console.log(c1.cname); // 访问变量  
c1.about(); // 调用方法  
  
let c2 = new City('G港', 5);  
console.log(c2.cname); // 访问变量  
c2.about();
```



类 - 批量创建对象

- 创建对象 (构造函数+new)

```
function City(cName, cLevel) {  
    this.cname = cName;  
    this.clevel = cLevel;  
}  
  
City.prototype.about = function () {  
    console.log(`兄嘚, 你跳【${this.cname}】~此地危险系数为: 【${this.clevel}】`);  
}
```

类 - 批量创建对象

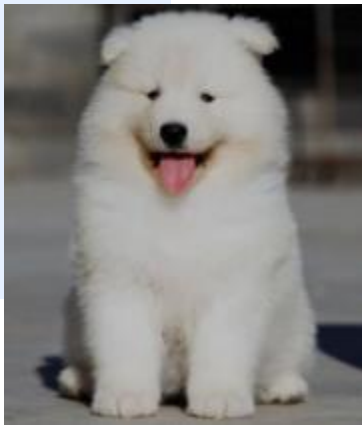
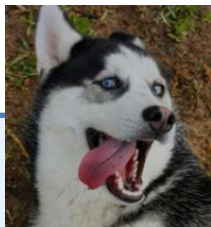
- 创建对象 (类class - TS)

```
class City{  
  cname:string; ← 成员变量: 定义在 类中  
  clevel:number;  
  
  constructor(cName:string, cLevel:number) {  
    this.cname = cName; ← 构造函数: 做初始化  
    this.clevel = cLevel;  
  }  
  
  about() {  
    console.log(`兄嘚, 你跳【${this.cname}】~此地危险系数为: 【${this.clevel}】`);  
  } ← 成员方法: 定义在 类中  
}
```

- 调用

```
let c1 = new City('P城', 1);  
console.log(c1.cname); // 访问变量  
c1.about(); // 调用方法
```

```
function City(cName, cLevel) {  
  this.cname = cName;  
  this.clevel = cLevel;  
}  
  
City.prototype.about = function () {  
  console.log(`兄嘚, 你跳【${this.cname}】~此地危险系数为: 【${this.clevel}】`);  
}
```





3

类

```
function City(cName, cLevel) {  
    this.cName = cName;  
    this.cLevel = cLevel;  
}  
  
City.prototype.about = function () {  
    console.log(`兄嘚，你跳【${this.cName}】~此地危险系数为：【${this.cLevel}】`);  
}
```

- ◆ 重温了 传统构造函数 批量创建对象 语法
- ◆ 掌握 class 创建类
- ◆ 掌握 class 中的 成员变量、构造函数、成员方法

```
class City {  
    cName: string;  
    cLevel: number;  
  
    constructor(name: string, level: number) {  
        this.cName = name;  
        this.cLevel = level;  
    }  
  
    about() {  
        console.log(`欢迎来到${this.cName}，此地危险系数为：${this.cLevel}`);  
    }  
}
```

课程安排

1

TS安装运行

2

核心语法



3

数据类封装

4

综合应用

3

数据类封装



LocalStroage 操作

DataHelper 类 设计

DataHelper 类 实现

LocalStorage 操作

- localStorage 用于 在浏览器端 持久化保存 键值对 数据



- localStorage 特点
 - 大小限制：5M (chrome) 更大数据 可以使用 浏览器本地数据库(indexDB 或 webSql)
 - 受同源访问限制，不允许跨域访问
 - 在浏览器 隐私模式 下无法使用
 - 因为在本地保存，不会发送数据，网络爬虫无法抓取
 - 只能存放字符串** 如果要存对象，可以使用 JSON 字符串

LocalStorage 操作

- localStorage 基本语法

方法名	作用
localStorage.setItem('key','value')	存放 键值对 数据
localStorage.getItem('key')	<u>根据 key 查询 value 值, 没有则返回 null</u>
localStorage.removeItem('key')	根据 key 删除 对应 键值对
localStorage.clear()	清空所有 键值对 数据

- localStorage 读写 对象

保存:

```
// 1.先将 对象 转成 JSON字符串, 然后再保存
let strJson: string = JSON.stringify( 对象 );

// 2.保存 json字符串 到 本地
localStorage.setItem('key', strJson);
```

读取:

```
// 1.取出 json字符串
let strJson: string | null = localStorage.getItem('key');

// 2.将 json字符串 转成 对象
let obj = JSON.parse(strJson as string);
```

3

数据类封装



LocalStroage 操作



DataHelper 类 设计

DataHelper 类 实现

业务需求

- 业务：评论操作
 - 加载评论列表 - 从本地 读数据 显示
 - 新增评论 - 存入 本地 和 页面
 - 删除评论 - 从 本地 和 页面 删除

【传智播客·黑马程序员】随着5G的逐步兴起，在 IT 领域的创新 将掀起 新的浪潮，快来学习吧~~！

【江苏传智专修学院】不是每位少年都适应应试教育，不服的话，那就来传智大学，没有应试，只有实用~用你的拼搏和不服输，让周围的白眼和同情都死开，让那份你珍惜的爱情拥有更坚实的基础~！

评论：

感激在传智日子，流下多少汗水，就免去了多少泪水~~~开心~！

X

兄啊，我在传智等你~~~！

X

发表

Storage Application Network Performance Memory Security Audits

Local Storage file:/// Session Storage IndexedDB Web SQL Cookies

Filter

```
[[{"content": "感激在传智日子，流下多少汗水，就免去了多少泪水~~~开心~！", "id": 1}, {"content": "兄啊，我"}]]
0: {"content": "感激在传智日子，流下多少汗水，就免去了多少泪水~~~开心~！", "id": 1}
1: {"content": "兄啊，我在传智等你~~~！", "id": 2}
```

DataHelper 类设计

- DataHelper 类设计
 - 加载评论列表时 -> readData() 方法
 - 新增评论 -> addData() 和 saveData() 方法

DataHelper
<u>dataKey</u> - localStrogae的键
<u>primaryKey</u> - 数据项主键名称
constractor- 构造函数

addData('评论内容') -> let list = readData() -> let pl = {content : '评论内容', id : 1} -> list.push(pl)
-> saveData(list)

The screenshot shows the Chrome DevTools Storage Inspector. On the left, the 'Storage' panel is open, showing 'Local Storage' with a file:// path. The main panel displays the 'Application' tab with a 'Filter' input. Below the filter, a table shows the 'Key' and 'Value' for the 'plData' key. The value is a JSON array of objects, each containing 'content' and 'id' properties. The objects represent comments stored in the local storage.

Key	Value
plData	[{"content": "感谢在传智日子，流下多少汗水，就免去了多少泪水~~~开心~!", "id": 1}, {"content": "兄嘞，我在传智等你~~~!", "id": 2}, ...]

Expanded view of the array:

- 0: {content: "感谢在传智日子，流下多少汗水，就免去了多少泪水~~~开心~!", id: 1}
- 1: {content: "兄嘞，我在传智等你~~~!", id: 2}
- 2: {content: "深圳黑马创维校区，老师都好风趣哦~~~", id: 3}

DataHelper 类设计

- DataHelper 类设计
 - 加载评论列表时 -> readData() 方法
 - 新增评论 -> addData() 和 saveData() 方法
 - 删除评论 -> removeDataById() 方法

DataHelper
<u>dataKey</u> - localStrogae的键
<u>primaryKey</u> - 数据项主键名称
constractor- 构造函数
readData - 读取本地数据,返回数组
addData - 新增数据对象,返回自动生成的id
saveData - 存入本地数据
removeDataById - 根据id 删除数据对象,返回 布尔值

removeDataById(1) -> let list = readData() -> list.splice方法
-> saveData(list)



The screenshot shows the Chrome DevTools Storage tab. On the left, the 'Local Storage' section is expanded, showing a file:// path. The main pane displays a table with 'Key' and 'Value' columns. The 'Key' column shows 'plData'. The 'Value' column shows a JSON array: `[{"content": "感谢在传智日子, 流下多少汗水, 就免去了多少泪水~~~开心~!", "id": 1}, {"content": "兄嘞, 我在传智等你~~~"}]`. Below the table, the expanded view of the array is shown, listing three objects with their 'content' and 'id' properties.

Key	Value
plData	[{"content": "感谢在传智日子, 流下多少汗水, 就免去了多少泪水~~~开心~!", "id": 1}, {"content": "兄嘞, 我在传智等你~~~"}]

▼ [{"content": "感谢在传智日子, 流下多少汗水, 就免去了多少泪水~~~开心~!", "id": 1}, {"content": "兄嘞, 我在传智等你~~~", "id": 2}, ...]
 ▶ 0: {"content": "感谢在传智日子, 流下多少汗水, 就免去了多少泪水~~~开心~!", "id": 1}
 ▶ 1: {"content": "兄嘞, 我在传智等你~~~", "id": 2}
 ▶ 2: {"content": "深圳黑马创维校区, 老师都好风趣哦~~~", "id": 3}

3

数据类封装

LocalStroage 操作

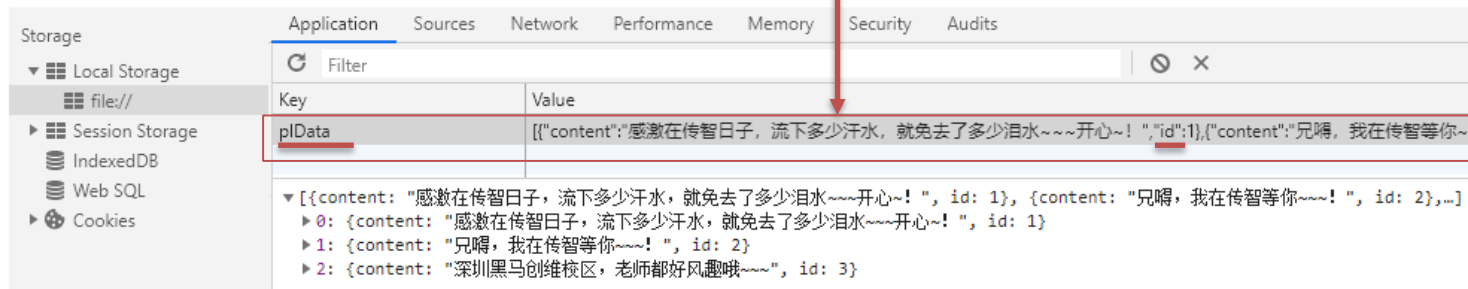
DataHelper 类 设计

DataHelper 类 实现



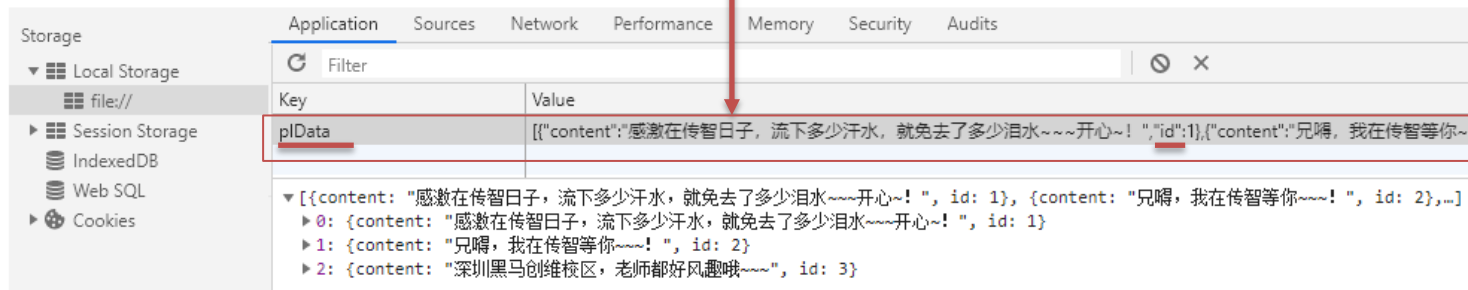
DataHelper 新增

- `addData(conStr: string)` 方法
 - 读取 `localStorage` 数据, 转成数组 `[{content:'内容1',id=1} ,...]`
 - 接收评论内容字符串 `conStr`
 - 将评论内容封装到对象, 并生成id `{ content:conStr , id=4}`
 - 将评论对象加入数组 `[{content:'内容1',id=1} ,..., { content:conStr , id=4}]`
 - 将数组转成字符串, 保存回 `localStorage`
 - 小细节: 返回 刚才生成的评论 `id` `"[{content:'内容1',id=1} ,..., { content:conStr , id=4}]"`



DataHelper 删除

- removeDataById(id: string) 方法
 - 读取 localStorage 数据, 转成数组 `[{content:'内容1',id=1} ,..., { content:conStr , id=4}]`
 - 找出数组中要删除的评论 (如: id=4)
 - 调用数组splice方法删除找出的对象 `[{content:'内容1',id=1} ,...]`
 - 将数组转成字符串, 保存回 localStorage
 - 小细节: 返回 bool 值 表示 删除结果 `"[{content:'内容1',id=1} ,...]"`





3

DataHelper

```
class DataHelper {
    dataKey: string;
    primaryKey: string;

    // 一、构造函数 --作用: 为对象 添加 各种属性-----
    constructor(dataKey: string, primaryKey: string) { ...
    }

    // 1.读取全部数据, 返回数组(如果没有读到数据 就返回 空数组)
    readData(): any { ...
    }

    // 2.存入本地数据 -----
    saveData(arrData: Array<Object>): void { ...
    }

    // 3.新增数据-----
    addData(conStr: string): number { ...
    }

    // 4.删除数据 -----
    removeDataById(id: string | number): boolean { ...
    }
}
```

◆ 完成 DataHelper 类的设计 与 实现

◆ 通过 DataHelper 协助评论案例的数据操作

【传智播客·黑马程序员】随着5G的逐步兴起, 在 IT 领域的创新 将掀起 新的浪潮, 快来学习吧~~!

【江苏传智专修学院】不是每位少年都适应应试教育, 不服的话, 那就来传智大学, 没有应试, 只有实用~
用你的拼搏和不服输, 让周围的白眼和同情都死开, 让那份你珍惜的爱情拥有更坚实的基础~!

评论:

感激在传智日子, 流下多少汗水, 就免去了多少泪水~~~开心~!

X

发表

课程安排

1

TS安装运行

2

核心语法

3

数据类封装

4

综合应用



Vue + Vuex + TS



4

综合应用



搭建 vue+ts 脚手架

项目结构 分析

项目代码 实现

Vue + Vuex + TS



搭建 vue+ts 脚手架

- 介绍

未来的前端发展，是逐步挺进企业级的开发业务，强类型的 typescript 已经广泛普及使用起来。

- 安装项

操作	操作路径	命令
安装 vue/cli	path	npm i -g @vue/cli
创建 vue 项目	path	vue create hmmemo
安装 vue/typescript	path/ hmmemo	vue add @vue/typescript
安装 vuex	path/ hmmemo	npm i vuex

```
> node_modules
> public
✓ src
  > assets
  > components
  > model
  > store
  ✓ App.vue
  TS main.ts
  TS shims-tsx.d.ts
  TS shims-vue.d.ts
  babel.config.js
  package-lock.json
  package.json
  README.md
  tsconfig.json
```

代码对比

```
? Use class-style component syntax? Yes
? Use Babel alongside TypeScript (required for modern mode, auto-detected polyfills, transpiling JSX)? Yes
? Convert all .js files to .ts? Yes
? Allow .js files to be compiled? Yes
```

App.vue - vuejs 版

```
<script>
import HelloWorld from './components/HelloWorld.vue'

export default {
  name: 'App',
  components: {
    HelloWorld
  }
}
</script>
```

App.vue - vuejs+TS 版

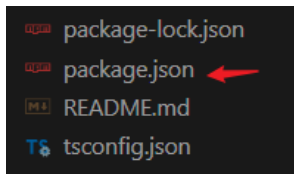
```
<script lang="ts">
import { Component, Vue } from 'vue-property-decorator';
import HelloWorld from './components/HelloWorld.vue';

@Component({
  components: {
    HelloWorld,
  },
})
export default class App extends Vue {}
</script>
```

搭建 vue+ts 脚手架

- 关闭 变量 未使用检查

eslint 会对声明 但未使用的变量做检查，如果发现会报错，造成不必要的麻烦



```
package.json ×
30   },
31   "eslintConfig": {
32     "root": true,
33     "env": {
34       "node": true
35     },
36     "extends": [
37       "plugin:vue/essential",
38       "eslint:recommended",
39       "@vue/typescript"
40     ],
41     "parserOptions": {
42       "parser": "@typescript-eslint/parser"
43     },
44     "rules": {
45       "no-unused-vars": 0
46     }
47   },
```


搭建 vue+ts 脚手架

操作	操作路径	命令
安装 vue/cli	path	npm i -g @vue/cli
创建 vue 项目	path	vue create hmmemo
安装 vue/typescript	path/ hmmemo	vue add @vue/typescript
安装 vuex	path/ hmmemo	npm i vuex

- ◆ 完成 vue/ts 项目的搭建
- ◆ 关闭了变量未使用 检查

4

综合应用



搭建 vue+ts 脚手架

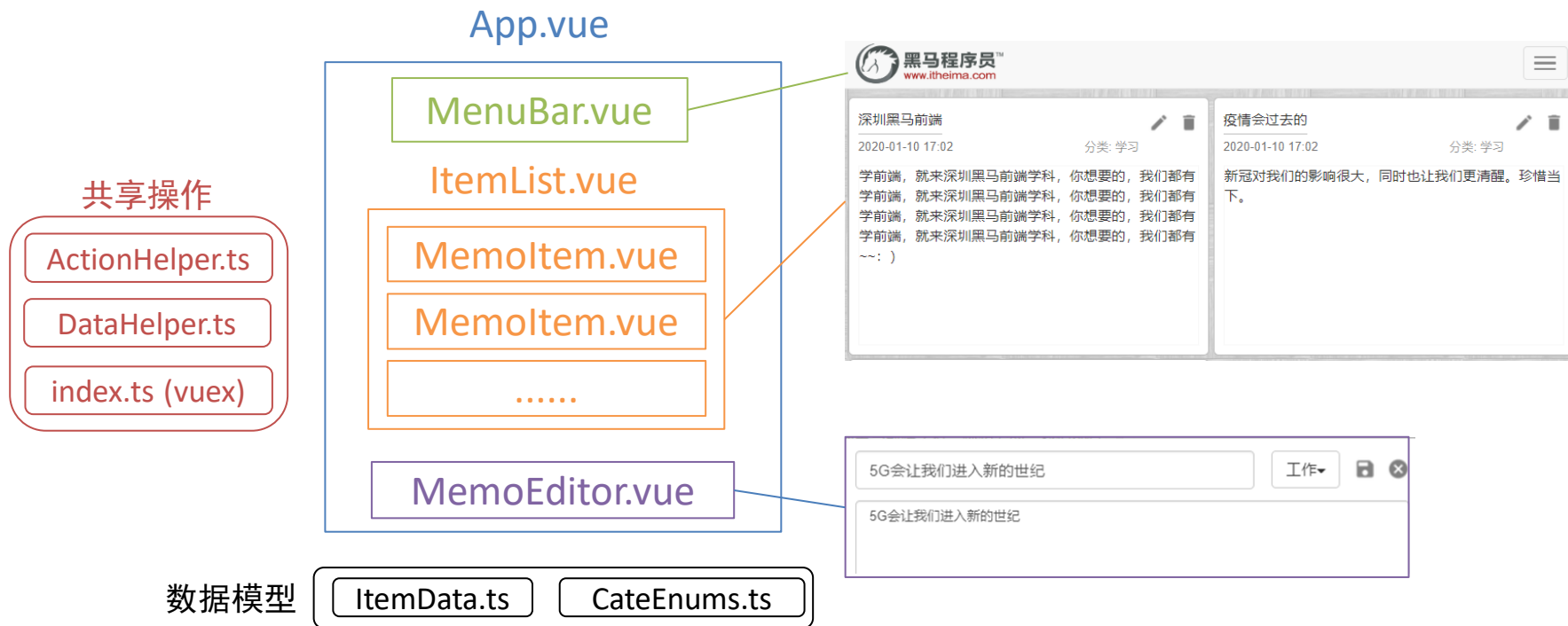
项目结构 分析

项目代码 实现

Vue + Vuex + TS



模型结构分析



项目结构分析

App.vue

MenuBar.vue

ItemList.vue

Memoltem.vue

Memoltem.vue

.....

MemoEditor.vue

```
> node_modules
✓ public
  > imgs
    common.css
    index.html
  > src
    > assets
    ✓ components
      ✓ ItemList.vue
      ✓ MemoEditor.vue
      ✓ Memoltem.vue
      ✓ MenuBar.vue
    ✓ model
      TS CategoryEnum.ts
      TS ItemData.ts
    ✓ store
      TS ActionHelper.ts
      TS DataHelper.ts
      TS index.ts
      ✓ App.vue
      TS main.ts
      TS shims-tsx.d.ts
      TS shims-vue.d.ts
    .gitignore
    babel.config.js
    package-lock.json
    package.json
    README.md
    tsconfig.json
```

数据模型

ItemData.ts

CateEnums.ts

共享操作

ActionHelper.ts

DataHelper.ts

index.ts (vuex)

项目结构分析

- ◆ 模型结构
- ◆ 文件结构

4

综合应用

搭建 vue+ts 脚手架

项目结构 分析

项目代码 实现



Vue + Vuex + TS



项目代码 实现-步骤

• 实现步骤

1. 实现整体展示

菜单条 - MenuBar.vue

笔记列表 - ItemList.vue
Menuitem.vue

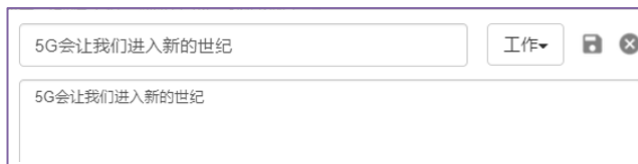
2. 实现新增

编辑框 - MenuEditor.vue

3. 实现删除

4. 实现编辑

5. 按分类显示笔记



LocalStorage

```
memoData [{"id":1,"categoryId":1,"title":"伊朗导弹袭击美军基地","content":"有点牛，相当牛，没文化，还是一个字：牛！","createTi..."}]
▼ [{id: 1, categoryId: 1, title: "伊朗导弹袭击美军基地", content: "有点牛，相当牛，没文化，还是一个字：牛！", ...}]
  ▶ 0: {id: 1, categoryId: 1, title: "伊朗导弹袭击美军基地", content: "有点牛，相当牛，没文化，还是一个字：牛！", ...}
  ▶ 1: {id: 2, categoryId: 1, title: "深圳黑马前端学科", content: "有趣的老师 + 丰富的案例 + 拼搏的氛围 = 走上人生巅峰", ...}
  ▶ 2: {id: 3, categoryId: 0, title: "来吧，兄囡，黑马深圳前端带你飞", content: "来吧，兄囡，黑马深圳前端带你飞", ...}
```

项目代码 实现-列表

• 列表 实现思路

- 创建 DataHelper.ts 和 ActionHelper.ts
- 通过 Vuex 共享 DataHelper 对象
- 创建 ItemList.vue - 通过 vuex store 找到 actionHelper 获取数据
- 创建 MenuItem.vue - 通过 父组件 获取 笔记数据

数据模型

ItemData.ts

CateEnums.ts

LocalStorage

```
memoData [{"id":1,"categoryId":1,"title":"伊朗导弹袭击美军基地","content":"有点牛，相当牛，没文化，还是一个字：牛！","createT..."}]
▼ [{"id":1, categoryId:1, title:"伊朗导弹袭击美军基地", content:"有点牛，相当牛，没文化，还是一个字：牛！", ...}]
  ► 0: {"id":1, categoryId:1, title:"伊朗导弹袭击美军基地", content:"有点牛，相当牛，没文化，还是一个字：牛！", ...}
  ► 1: {"id":2, categoryId:1, title:"深圳黑马前端学科", content:"有趣的老师 + 丰富的案例 + 拼搏的氛围 = 走上人生巅峰"}
  ► 2: {"id":3, categoryId:0, title:"来吧，兄囡，黑马深圳前端带你飞", content:"来吧，兄囡，黑马深圳前端带你飞", ...}]
```



共享操作

ActionHelper.ts

DataHelper.ts

index.ts (vuex)

ItemList.vue

Memoltem.vue

Memoltem.vue

.....

修改方法-思路

memoList 数组

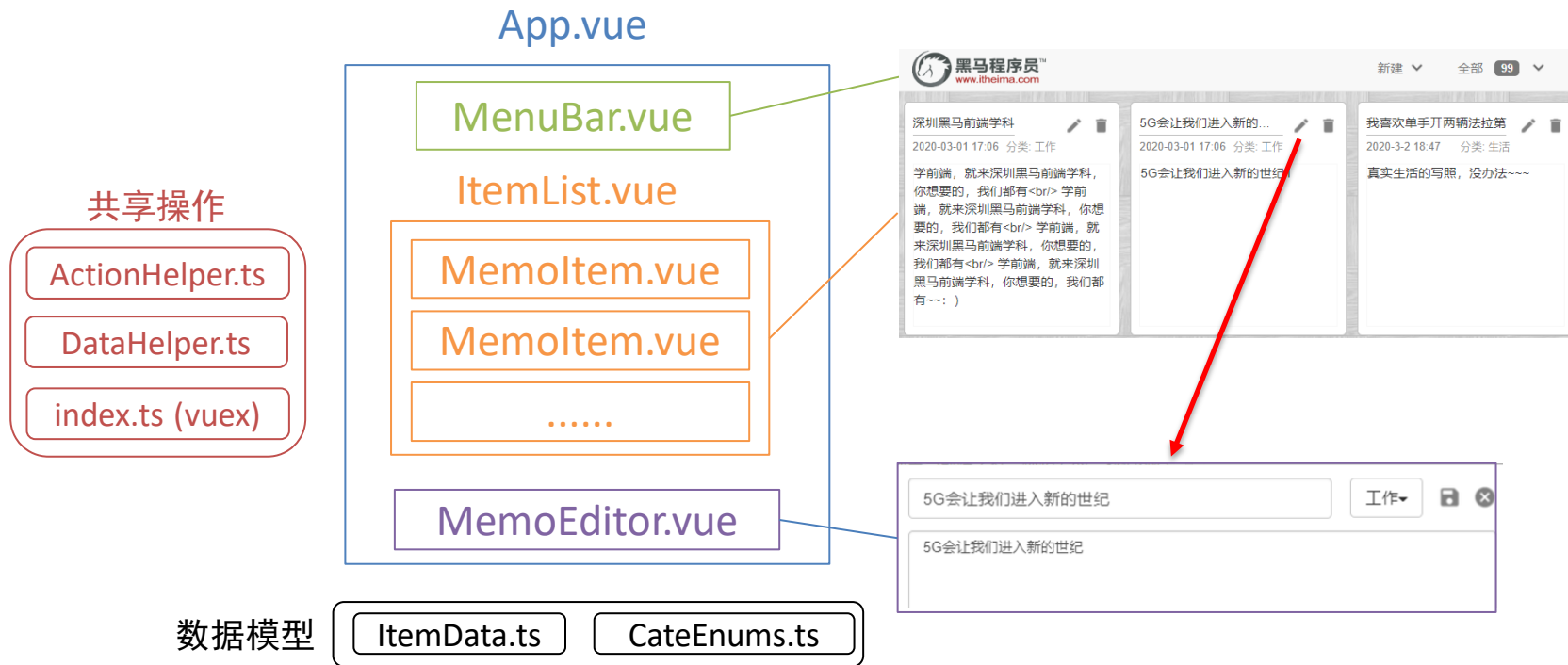
```
0: {id: 1, categoryId: 1, title: "伊朗导弹袭击美军基地", content: "有点牛，相当牛，没文化，还是一个字：牛！",...}  
1: {id: 2, categoryId: 1, title: "深圳黑马前端学科", content: "有趣的老师 + 丰富的案例 + 拼搏的氛围 = 走上人生巅峰",...}  
2: {id: 3, categoryId: 0, title: "来吧，兄嘚，黑马深圳前端带你飞", content: "来吧，兄嘚，黑马深圳前端带你飞",...}
```

要更新的 对象

```
{id: 3, categoryId: 1, title: "讨厌", content: "死鬼~~讨厌~~~",...}
```

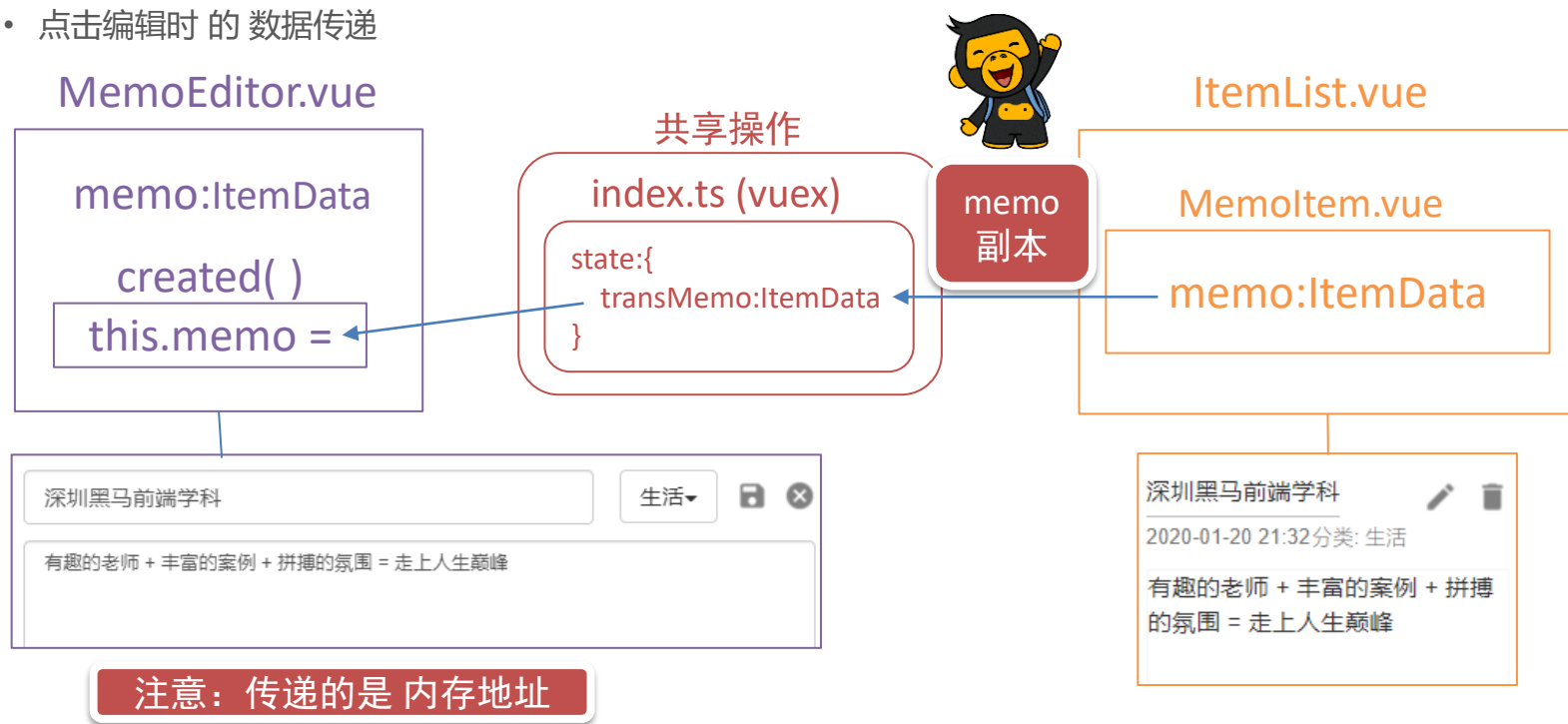
▼ Local Storage

项目代码 实现-修改



项目代码 实现-编辑

- 点击编辑时的 数据传递



删除方法-思路

memoList 数组

```
0: {id: 1, categoryId: 1, title: "伊朗导弹袭击美军基地", content: "有点牛，相当牛，没文化，还是一个字：牛!",...}  
1: {id: 2, categoryId: 1, title: "深圳黑马前端学科", content: "有趣的老师 + 丰富的案例 + 拼搏的氛围 = 走上人生巅峰",...}  
2: {id: 3, categoryId: 0, title: "来吧，兄嘚，黑马深圳前端带你飞", content: "来吧，兄嘚，黑马深圳前端带你飞",...}
```

要删除对象的id

id: 3

▼ Local Storage

项目代码 实现-删除

删除 实现思路

a.在 MenuItem.vue 中

a1.在 class 中 添加 doDel 方法，并调用 **ActionHelper** 的删除方法

a2.为 删除 按钮添加 点击事件，绑定 doDel 方法

AcrtionHelper类中的删除方法

```
//2.3 删除笔记
remove(id: number): void {
  //a.根据id 找出 要删除的 对象 在数组中的 下标
  let index: number = this.memoList.findIndex((ele) => {
    return ele.id === id;
  })

  //b.根据下标 调用 数组.splice方法来删除对象
  if (index > -1) {
    this.memoList.splice(index, 1);
    //c.将删除对象后的 数组重新保存回 localStorage
    this.dataHelper.saveData(this.memoList);
  }
}
```



共享操作

ActionHelper.ts

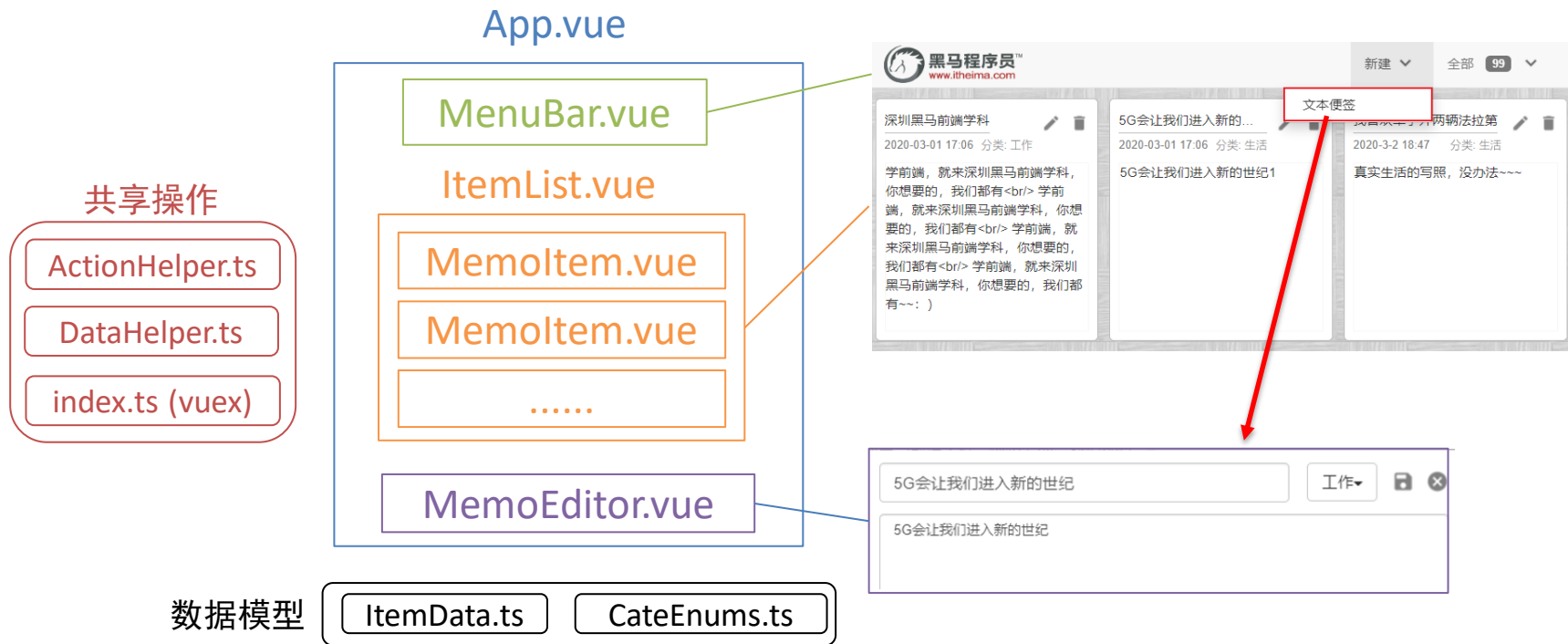
DataHelper.ts

index.ts (vuex)

LocalStorage

memoData	[[{"id":1,"categoryId":1,"title":"伊朗导弹袭击美军基地","content":"有点牛，相当牛，没文化，还是一个字：牛！","createTi...
▼	[[{"id": 1, categoryId: 1, title: "伊朗导弹袭击美军基地", content: "有点牛，相当牛，没文化，还是一个字：牛！","-"},...]]
▶0:	{id: 1, categoryId: 1, title: "伊朗导弹袭击美军基地", content: "有点牛，相当牛，没文化，还是一个字：牛！","-"},...}
▶1:	{id: 2, categoryId: 1, title: "深圳黑马前端学科", content: "有趣的老师 + 丰富的案例 + 拼搏的氛围 = 走上人生巅峰",...}
▶2:	{id: 3, categoryId: 0, title: "来吧，兄囑，黑马深圳前端带你飞", content: "来吧，兄囑，黑马深圳前端带你飞","-"},...}

项目代码 实现-新增



项目代码 实现-编辑

- 点击编辑时的 数据传递

MemoEditor.vue

memo:ItemData

created()

this.memo =

memo
副本

共享操作

index.ts (vuex)

```
state:{  
  transMemo:ItemData  
}
```

ItemList.vue

MemolItem.vue

memo:ItemData

深圳黑马前端学科

生活

有趣的老师 + 丰富的案例 + 拼搏的氛围 = 走上人生巅峰

深圳黑马前端学科

2020-01-20 21:32分类: 生活

有趣的老师 + 丰富的案例 + 拼搏
的氛围 = 走上人生巅峰

注意：传递的是 内存地址

项目代码 实现-问题

- 点击编辑时的 数据传递

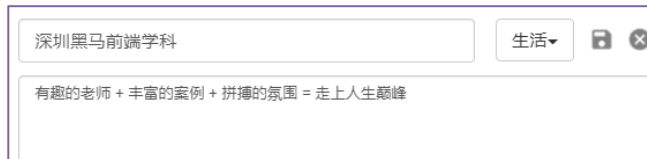


new ItemData()

```
{
  "id":-1,
  "categoryId":-1,
  "title": "",
  "content": "",
  "createTime": ""
}
```

MemoEditor.vue

```
memo:ItemData
created( )
this.memo =
```



memo:ItemData

```
{
  "id":5,
  "categoryId":1,
  "title":"有趣的老师...",
  "content":"真实生活的写照...",
  "createTime":"2020-2-21..."
}
```


项目代码 实现-统计



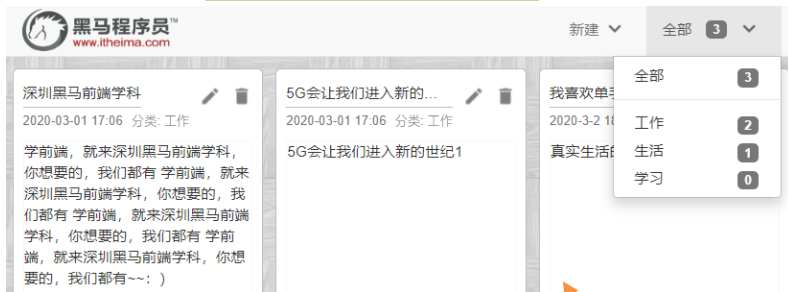
memoList:Array<ItemData>

共享操作 store

```
state:{  
  aHelper:ActionHelper  
}
```

项目代码 实现-筛选

MenuBar.vue



共享操作 store
memoList:Array<ItemData>

```
state:{  
  aHelper : ActionHelper,  
  filterCatId : -1  
}
```

ItemList.vue

根据 filterCatId 来生成列表

项目代码 实现



- ◆ 菜单条 - MenuBar.vue
- ◆ 笔记列表 - ItemList.vue + MenuItem.vue
- ◆ 实现新增
- ◆ 实现删除 - 编辑框 MenuEditor.vue
- ◆ 按分类显示笔记

